Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 44 (2016) 341 – 346

www.elsevier.com/locate/procedia

6th CIRP Conference on Assembly Technologies and Systems (CATS)

Towards Energy Optimization using Trajectory Smoothing and Automatic Code Generation for Robotic Assembly

Daniel Gleeson[a,b,*], Staffan Björkenstam[a], Robert Bohlin[a], Johan S. Carlson[a], Bengt Lennartson[b]

[a]*Geometry and Motion Planning, Fraunhofer-Chalmers Centre, Chalmers Science Park, SE-412 88 Göteborg, Sweden*

[b]*Automation Research Group, Department of Signals and Systems, Chalmers University of Technology, SE-412 96 Göteborg, Sweden*

[*] Corresponding author. Tel.: +46 (0)31-772 4243; fax: +46 (0)31-772 4260. *E-mail address:* daniel.gleeson@fcc.chalmers.se

Abstract

In automated industrial production, the efficiency of robotic motions directly affects both the final throughput and the energy consumption. By simulating and optimizing robot trajectories, cycle times and energy consumption can be lowered, or redundant robots can be detected. Here a polynomial basis function trajectory parametrization is presented, which enables direct export to executable robot code, and reduces the number of variables in the optimization problem. The algorithm finds time-optimal trajectories, while including collision avoidance and fulfilling joint, velocity and acceleration limitations. Applied torques are used as an approximation of the energy consumption to analyse the smooth trajectories, and successful tests show potential reductions of 10% for a standard industrial robot stud welding station.

Peer-review under responsibility of the organizing committee of the 6th CIRP Conference on Assembly Technologies and Systems (CATS)

Keywords: Robotics; Motion Planning; Optimal Control; Industrial Production

1. Introduction

In manufacturing industries using industrial robots and having a high level of automation, e.g. the automotive industry, the setup of an assembly line can be a highly complex task. It has to be performed every time a new product is to be produced, and when changes are made to the production line. By automating as much as possible of the setup phase, new products can come faster into production, and factory down-time can be reduced. Using optimization, the cycle times of the robots in each work station can be lowered, and there is an opportunity to also reduce the energy consumed.

Both in academia and in industry, the problem of simplifying the generation of assembly line robot code [1], and creating flexible production lines [2] has received a lot of attention. State of the art methods of today are helping process engineers finding short and fast robot trajectories, solving high dimensional path planning [3,4], scheduling [5,6] and workload distribution [7] problems. In academia more robust and effective algorithms are being developed [8].

The workflow in industry can still be further improved by improving the trajectories and removing any manual steps required. An example of such manual step was addressed in paper [9], where the goal was to remove the manual task of choosing zone radii for a given piecewise linear trajectory with via points. The problem was set up using variables that can be used directly as parameters in the set of available robot controller functions, which makes it possible to directly export the solutions to robot code. Only part of the available variable freedom was used in the optimization, since the initial via points were fixated and the only parameters that affected the geometrical shape were the zone radii.

The contribution of this work is to further improve the solutions by using non-fixated via points, giving the optimization algorithm a larger search space. This is achieved by reparameterizing the problem using piecewise polynomial functions, improving the robustness of the trajectory parametrization. The limited number of control variables available as robot controller commands is still used, so that solutions can be directly exported to robot code. An approximation of the energy consumed by the robot is also used to study the potential of offline energy optimization of robot trajectories.

2. Method

The method and optimization algorithm presented in this paper is a development and reformulation of the work presented in Gleeson et al.[9], which in turn is largely based on the ideas presented in Björkenstam et al.[10]. Summarizing the contin-

Peer-review under responsibility of the organizing committee of the 6th CIRP Conference on Assembly Technologies and Systems (CATS)
doi:10.1016/j.procir.2016.02.099

uous problem formulation in general terms, we have an optimal control problem (1) of finding the control signal u, which minimizes the cost functional J, composed of initial and final costs Φ, as well as running costs described by the Lagrangian L. Furthermore, the control and state should fulfill dynamic constraints (1b), as well as equality (1c) and inequality (1d) constraints.

$$\min_u J = \Phi(x(t_a), t_a, x(t_b), t_b) + \int_{t_a}^{t_b} L(x(t), u(t), t)dt \quad (1a)$$

$$\text{such that} \quad \dot{x}(t) = f(x(t), u(t), t) \quad (1b)$$

$$g(x(t), u(t), t) \geq 0 \quad (1c)$$

$$H(x(t_a), t_a, x(t_b), t_b) = 0 \quad (1d)$$

for $t \in [t_a, t_b]$.

The discretization method and trajectory parametrization used in [9] is also used here, but the problem is reformulated to decrease the number of variables and make the problem easier for the optimization algorithm. Since an interior point algorithm, the Ipopt solver developed by Wächter and Biegler[11], is used to solve the resulting optimization problem, a feasible initial guess will in general reduce the number of steps and the time it takes to convergence to an optimal solution. The variables in the new formulation are more physical and easier to use when an initial feasible point is to be set up. The initial point makes use of the solution given by the path planning algorithm developed by Bohlin and Kavraki[4], which is a piecewise linear collision free trajectory.

The benefit of using this trajectory parametrization is that the reduced convergence issues make it possible to relax the constraints on the via points and allow the optimizer to use a larger part of the search space. With this increased flexibility comes also the possibility of considering other objective functions. Time optimization is used here, retaining the correspondence to the trajectories produced by the robot controllers of today. An energy consumption model is used to compare the solutions to each other and give an indication of how large the potential is for energy reductions. To include energy optimization would require an outer optimization loop, and this will be the focus of future work. Still, optimizing with respect to time will smooth the trajectory, giving noticeable energy reduction.

2.1. Parametrizing the trajectory

To be able to generate robot code for a specific trajectory, a number of variables will have to be defined to specify the robot path. The overall structure of a robot path is defined by its initial point q_{start}, final point q_{end} and via points it should reach between them. These points are vectors of joint values, with typically six joints for a standard industrial robot. The via point joint vectors are denoted q_{mid} as they define the midpoint of a via point zone. For each via point, a zone radius defines an area where the robot is allowed to deviate from the otherwise piecewise linear path, smoothing out sharp corners and making it possible to maintain a velocity through the transition between linear segments.

A simplified sketch of a robot trajectory in joint space can be seen in Fig. 1 along with some notations used to describe

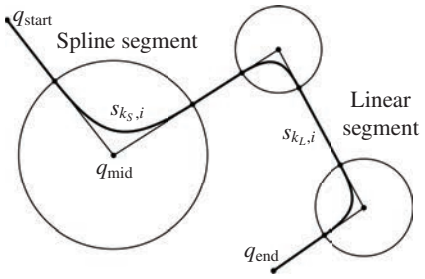
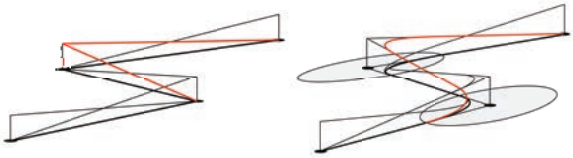


Fig. 1: Notations for linear and spline segments along the trajectory. The parametrization parameters, $s_{k,i}$ define N positions in each segment.



(a) Linear trajectory (b) Trajectory with via point zones.

Fig. 2: The two figures show the trajectory and corresponding polynomials for a linear trajectory and a spline trajectory defined by via point zones. For the spline trajectory the corresponding polynomials of each via point is non zero within neighbouring via point zones.

the variables and different parts of the trajectory. Each segment is parametrized by a parameter in a unit length interval $s_{k,i} \in [k, k+1]$. For the seven segments of the trajectory $k \in \{0, ..., 6\}$ seen in the figure, variables are specifically shown for the spline segment $k_S = 1$ and the linear segment $k_L = 4$. The shape of robot trajectories used here is the same as was used in [9], which have been found to correspond very well to the interpolated joint trajectories used by ABB robot controllers. Similar trajectories are also used by other industrial robot manufacturers (e.g. KUKA) even if there are minor differences.

In [9], the parametrization of the trajectory was divided into linear phases and spline phases, and the starting point and end point of each phase had to be defined and linked to the neighbouring phases. Points along the trajectory within each phase are then defined using these starting points and end points, as well as the via points of the spline phases. But as previously stated, it is only the via points and zone sizes that define the shape of the path. Here we instead parameterize the trajectory directly from these variables without explicitly defining the coordinates of the transitions between linear segment and spline segment. In order to do this we have to specify how each via point affects the position of points along the trajectory by finding and composing the polynomials that make up the trajectory. The parametrization of the trajectory will then consist of a summation over via point-vectors q_i and corresponding polynomial functions p_i:

$$q(s) = \sum_i q_i p_i. \quad (2)$$

The polynomial $p_i(s, r_A, r_B, \ell)$ will be a function of the trajectory parameter s, the zone sizes r_A and r_B, and the distance

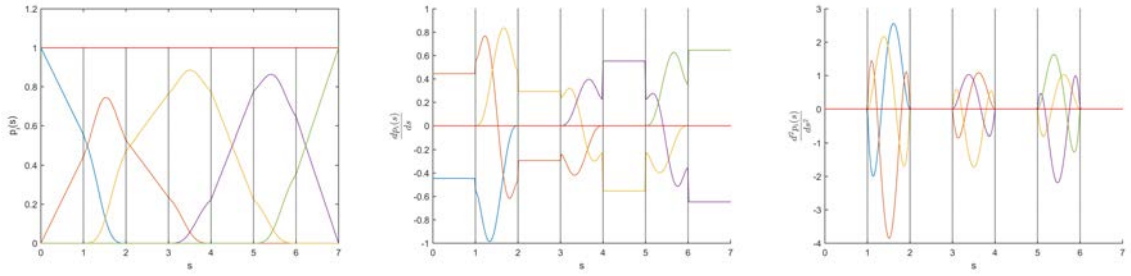


Fig. 3: Plot of the polynomials for each via point along with first and second derivatives for a simple example trajectory.

between via points ℓ.

These polynomials would in the case of piecewise linear trajectories be triangular functions, see Fig. 2a. To obtain a point along the trajectory is in the linear case a matter of taking a weighted sum of the two nearest via points, and no via point will affect the final trajectory further away than its nearest neighbouring via points. Looking at a plot of the piecewise linear polynomials and the support of the triangular basis functions, we have at most two non-zero functions at any given parameter value.

Considering a trajectory with via point zones, the trajectory can still be formulated in the form of Eq. (2) with a polynomial defining each segment. Now via points will affect the trajectory for points within its neighbouring zones and the polynomial will be non-zero for a larger span of parameter values, with up to three via points affecting the position of the trajectory at a given parameter value.

This means that each via point joint vector has its corresponding piecewise polynomial, p_i, which will be non-zero over a maximum of five segments. Enumerating these segments with roman numerals we have the following expressions:

$$p_i(s) = \begin{cases} \dfrac{r_{B[i-1]}\widetilde{p}(s)s}{\ell_{[i-1]}} & \text{(I)} \\ \dfrac{r_{B[i-1]} + s(\ell_{[i-1]} - r_{B[i-1]} - r_{A[i]})}{\ell_{[i-1]}} & \text{(II)} \\ 1 - \dfrac{r_{A[i]}(1-\widetilde{p}(s))(1-s)}{\ell_{[i-1]}} - \dfrac{r_{B[i]}\widetilde{p}(s)s}{\ell_{[i]}} & \text{(III)} \\ 1 - \dfrac{r_{B[i]} + s(\ell_{[i]} - r_{B[i]} - r_{A[i+1]})}{\ell_{[i]}} & \text{(IV)} \\ \dfrac{r_{A[i+1]}(1-\widetilde{p}(s))(1-s)}{\ell_{[i]}} & \text{(V)} \end{cases}$$

where $\widetilde{p}(s)$ is a polynomial ensuring continuous higher derivatives at the segment boundaries for the trajectory. See the left plot in Fig. 3 to see how the polynomials connect to each other for a simple test problem.

Depending on the order of the polynomial $\widetilde{p}(s)$ it will assure first, second or third-order continuity of the trajectory at segment boundaries as described in [12] and [13]. The three implemented polynomials have the following form:

$$\begin{aligned} &\text{First order continuity:} && \widetilde{p}(s) = 3s^2 - 2s^3 \\ &\text{Second order continuity:} && \widetilde{p}(s) = 10s^3 - 15s^4 + 6s^5 \\ &\text{Third order continuity:} && \widetilde{p}(s) = 35s^4 - 84s^5 + 70s^6 - 20s^7 \end{aligned}$$

Higher derivatives of these polynomials with respect to the trajectory parameter s will also be used to set limits on the joint velocity and joint acceleration of the robot. The plot of the first and second derivative can be seen in the middle and right plot in Fig. 3. These will include higher derivatives of the polynomials $\widetilde{p}(s)$ but their computation is straight forward. Shown below are the expressions for the first-order derivatives for the five different segments.

$$\frac{\partial p_i}{\partial s} = \begin{cases} \dfrac{r_{B[i-1]}\left(\frac{\partial \widetilde{p}(s)}{\partial s}s + \widetilde{p}(s)\right)}{\ell_{[i-1]}} \\ \dfrac{\ell_{[i-1]} - r_{B[i-1]} - r_{A[i]}}{\ell_{[i-1]}} \\ \dfrac{r_{A[i]}\left(\frac{\partial \widetilde{p}(s)}{\partial s}(1-s) + (1-\widetilde{p}(s))\right)}{\ell_{[i-1]}} - \dfrac{r_{B[i]}\left(\frac{\partial \widetilde{p}(s)}{\partial s}s + \widetilde{p}(s)\right)}{\ell_{[i]}} \\ -\dfrac{\ell_{[i]} - r_{B[i]} - r_{A[i+1]}}{\ell_{[i]}} \\ -\dfrac{r_{A[i+1]}\left(\frac{\partial \widetilde{p}(s)}{\partial s}(1-s) + (1-\widetilde{p}(s))\right)}{\ell_{[i]}} \end{cases}$$

Second-order derivatives are obtained by an additional differentiation but are not included here. The general appearance of the polynomials can be seen in the right plot in Fig. 3.

Points along the trajectory can now be found by multiplying the via points with their respective polynomial, and summing according to Eq. (2). For a sequence of trajectory parameter values we get the trajectory as seen in Fig. 4.

2.2. Constraints

All constraints related to the shape of the trajectory as well as limits on joint values, joint velocities and joint accelerations are stated as functions including the via point polynomials $p_i(s)$, first and second-order derivatives seen in Fig. 3.

Upper and lower bounds for the joint velocities can be written as

$$\dot{q}_{lower} \leq \dot{q} \leq \dot{q}_{upper},$$

where

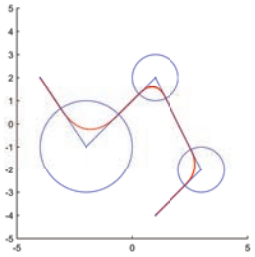


Fig. 4: The trajectory for the test case is then found by multiplying each via point with its corresponding polynomial for different values of the parameter s.

$$\dot{q} = \sum_i q_i \frac{\partial p_i}{\partial s}\frac{ds}{dt}. \tag{3}$$

Taking the time derivative of this expression gives a similar expression for limits on joint accelerations

$$\ddot{q}_{lower} \leq \ddot{q} \leq \ddot{q}_{upper},$$

where

$$\ddot{q} = \sum_i q_i \left(\frac{\partial^2 p_i}{\partial s^2}\left(\frac{ds}{dt}\right)^2 + \frac{\partial p_i}{\partial s}\frac{d^2 s}{dt^2}\right). \tag{4}$$

In both the expression for $\dot{q}$ (3), and $\ddot{q}$ (4), time derivatives of the parameter s are included. These time derivatives are approximated using numerical differentiation of s_i.

The expression describing points along the trajectory (2), is used in a similar way as the expressions for velocity and acceleration constraints to ensure that the solution trajectory is collision free. This is done by setting upper and lower bounds on the position for points along the trajectory, and we denote these constraints box-constraints [9]. The values for the constraints are found using a first order approximation of the distance to the surrounding geometry in joint space. The solution is then iteratively improved while maintaining a collision free trajectory. A more detailed explanation of this iterative procedure can be found in [10], where the points along the trajectory are not constrained to follow an implementable robot trajectory.

The trajectory parameters $s_{k,i}$ should increase from k to $k+1$ for each segment, as seen in the polynomial plot in Fig. 3. This is done by locking the first and last parameter value for each segment. For each segment k we have

$$s_{k,0} = k \qquad \text{and} \qquad s_{k,N-1} = k+1.$$

The parameter values should be monotonically increasing which means that the constraint $s_{k,i} < s_{k_i+1}$ is included for all segments k and all neighbouring points i.

The initial and final velocity is assumed to be zero, which is implemented as constraints on the trajectory parameters $s_{[0,0]} = s_{[0,1]} = 0$ and $s_{[N-1,n-2]} = s_{[N-1,n-1]} = N$. Over segment bounds the joint velocities should match. This is not automatically enforced by the parametrization even if the geometrical path is smooth. Looking at Fig. 3, which shows the derivatives of the trajectory polynomials, it can be noticed how the non-uniform time spacing between segments introduce discontinuities at the segment boundaries. The constraints implemented to keep the joint velocities continuous when transitioning from a linear segment to a spline segment take the following form:

$$t_S(\ell - r_A - r_B)(s_{L,[n-2]} - s_{L,[n-1]}) = t_L r_A (s_{S,[1]} - s_{S,[0]}). \tag{5}$$

Here a subscript L or S denotes the linear or spline segment, respectively. The radius r_A is taken to be the radius of the closest zone and r_B the radius of the neighbouring zone.

The variable ℓ used in (5) is the distance between via points, and is constrained to match this distance by including the following constraint:

$$\sum_j \left(q_{\text{mid }[i][j]} - q_{\text{mid }[i+1][j]}\right)^2 = (\ell_{[i]})^2. \tag{6}$$

When exporting a trajectory as RAPID code (a robot controller language for ABB robots), the size of a zone is determined by a zone radius, r_{TCP}. This radius describes a three dimensional sphere around the Tool Center Point (TCP) at each via-point. In [9] the fact that the via points were fixed meant that a good approximation of the joint space size of the zone could be used, but since the size of the zone in joint space varies in different directions, this approximation would not be as accurate when the via point positions are allowed to change. So instead of using an approximation in joint space, the TCP-coordinates are included as variables; x for positions at the boundary between a linear segment and a zone, and x_{mid} for the via points. This makes it possible to constrain the positions of the initial and final point of each zone to match the TCP-radius:

$$\sum_j \left(x_{\text{mid }[j]} - x_{[j]}\right)^2 = r_{TCP}^2. \tag{7}$$

Additional constraints would be needed linking each vector of joint values, q, to its corresponding TCP-value, x. The constraints include non-linear forward kinematic calculations that recursively calculate TCP-values from the joint coordinates. Using $f_{FK}(q)$ to denote this forward kinematics calculation of rotations and translations, the constraint can be expressed as:

$$f_{FK}(q) - x = 0. \tag{8}$$

Here q can be either the joint space values of a via point or a point along the trajectory at the zone boundary. In the latter case the joint values are calculated using (2).

Since the radius for a via point zone cannot overlap in TCP-space, if the solution is to be exported to RAPID code, we introduce constraints of the following kind, for each combination of neighbouring via point TCP-distances and TCP zone radii r_{TCP}:

$$\sum_j \left(x_{\text{mid }[i][j]} - x_{\text{mid }[i+1][j]}\right)^2 - r_{TCP}^2 \geq 0.$$

Even if explicit torque bounds are not added to the problem, we

can still look at the sum of squared torques (weighted with the non-uniform time step) to get an approximation of the energy consumption. The torques are calculated using inverse dynamics seen in (9) below, and the efficient recursive algorithm used is described by Featherstone[14].

$$\begin{aligned} Q &= f_{ID}(q(t), \dot{q}(t), \ddot{q}(t)) \\ &= M(q(t))\ddot{q}(t) + C(q(t), \dot{q}(t)) + G(q(t), \dot{q}(t)) \end{aligned} \tag{9}$$

Here $M(q(t))$ is the system's mass matrix, $C(q(t), \dot{q}(t))$ includes the centrifugal and Coriolis forces, and $G(q(t), \dot{q}(t))$ are the external forces including gravity.

The problem has now been discretized and set up as a general optimization problem with via points, zone radii, parameterization variables, and time durations of each segment uniquely specifying a trajectory. In the following section the optimization is carried out on a two dimensional case and an industrial case, and the performance and general appearance of the solutions are presented.

3. Results

By locking all but two of the robot joints, a simple test example can be set up. Using only two degrees of freedom means that the solution can be plotted in joint space. The same two-dimensional test case as was used in [9] is used here, where the second and third joint of the robot is free to move and the robot should move around some fixed geometry. To more clearly show how the possibility of moving the via points affects the trajectory, a more crude initial trajectory is used than the one found by the path planner. In Fig. 5 the joint space and in Fig. 6 the TCP space of the optimal solution with fixated via points is shown. The zone sizes in Fig. 6 clearly shows what finally limits the increasing zone sizes. The first zone is limited by the distance to the second via point, the zone radius can only be as large as half this distance. The second and third zone are increased as much as possible without overlapping, there is still considerable clearance to the surrounding geometry. Converting the circular TCP-zone into joint space will give complex geometries in higher dimensions, and even here, in only two dimensions, Fig. 5 shows how the zones are smeared out into non-obvious shapes.

In Fig. 7 the joint space and in Fig. 8 the TCP space of the optimal solution is shown when both via points and zone sizes are free variables. Here the zones are smaller, but the via points have moved closer to the geometry, and the resulting trajectory is both smoother and shorter. The limiting factor during the optimization is the distance to the geometry, and the found solution moves close to the surrounding geometry but maintains the specified clearance of 5 mm.

The usage of via points and zone radii as optimization variables makes it possible to export the final solution to ABB RAPID code. Running the robot code in Robot Studio gives final times that can be compared to further validate the solutions. The optimal solution when both via points and zone sizes are optimized is a trajectory that takes 3.5 s to run in Robot Studio. When the via points are fixed the zone size optimization algorithm smooths the trajectory, but the bad initial solution cannot be compensated for, giving a final time of 4.4 s.

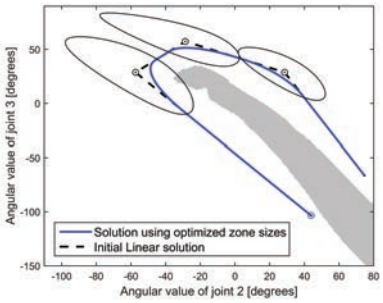


Fig. 5: Plot of solution joint values for the two-dimensional case where the zone sizes have been optimized, but via points kept fixed.

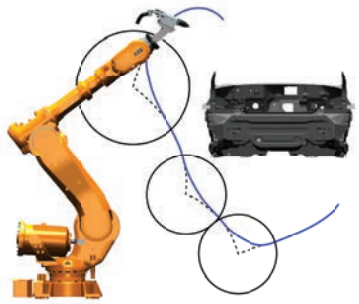


Fig. 6: Plot of the TCP trajectory for the solution to the two-dimensional case where the zone sizes have been optimized, but via points kept fixed.

Testing the algorithm on an industrial case, the trajectory for one of the robots in a stud welding station is optimized. The trajectory consists of 23 studs to be visited, starting and ending in a home position. The path planner finds an initial trajectory with 24 point-to-point movements, out of which four have additional via points included to avoid the surrounding geometry. For these four trajectories it is possible to optimize the position of the via points and the zone sizes. The final time as reported by Robot Studio, as well as the approximate energy reduction per joint is seen in Table 1. Worth noting when looking at the energy reduction values is that the highest torque values are found for joint 2 and 3, these are the ones that were free in the two-dimensional example and are the joints that need to hold the bulk of the robots weight. Since the initial solution is very good, we see that the additional freedom of being able to move the via points only give minor time reductions.

4. Conclusions

Using zone sizes and via points as variables in the optimization problem, fast and smooth collision free trajectories have been found. When a good initial trajectory is used the use of via points as free variables in the optimization problem might only give minor improvements. Table 1 shows a decrease of the final time from 12.00 s to 11.88 s. But the ability to change the via points is important when the optimization expands to include energy optimization. The assumption we make is that the time optimal solutions found mimics the internal workings of the robot controller. This means that energy optimization will

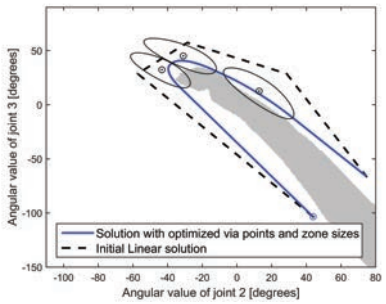


Fig. 7: Plot of the optimal trajectory for the two-dimensional example, where zone sizes and via points have been optimized.

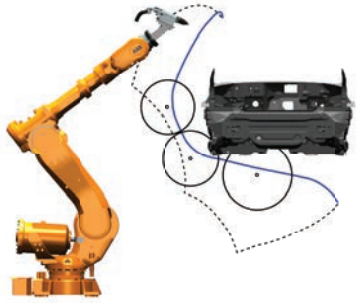


Fig. 8: Plot of the TCP trajectory for the solution of the two-dimensional case, where the zone sizes and via points have been optimized.

have to be performed in an outer optimization problem, if it still should be possible to export the solution directly to robot code.

Other methods have been used to optimize the energy consumption of industrial robots. In [15] the robot path is assumed to be fixed and the optimization is done on the velocity profile, with low level robot commands being used to precisely specify the position of the robot in each time step. In [16] the optimization is performed on-site by comparing collected data to a robot model, optimizing the trajectories and exporting the solution using custom robot commands. In [17] the collision avoidance is directly included in the optimization problem. Our use of box-constraint approximations avoids this, but requires iterative solving of the optimization problem.

Solving the energy optimization problem, using a time optimization subproblem as stated here, makes optimization possible without access to the physical robot, while maintaining a high degree of freedom for the optimizer to utilize.

Acknowledgements

This work was carried out at the Wingquist Laboratory VINN Excellence Centre within the Production Area of Advance at Chalmers University of Technology. It was supported by the Swedish Governmental Agency for Innovation Systems, the European Communitys Seventh Framework Programme (FP7/2007-2013, grant agreement No 609391) and the AREUS project.

Finally, we would like to thank Jonas Kressin and Simon Vajedi at FCC for their help with generating corresponding trajectories using ABB:s Robot Studio.

Table 1: Final time and energy consumption approximation for stud welding station.

	Velocity tuned	Optimal radii	Optimal
Time [s]	13.87	12.00	11.88
Time reduction [%]	-	−13.5 %	−14.3 %
Energy fraction [%]			
Joint 1	-	+0.7 %	+1.3 %
Joint 2	-	−3.5 %	−2.2 %
Joint 3	-	−9.4 %	−10.0 %
Joint 4	-	−9.6 %	−11.9 %
Joint 5	-	−9.7 %	−12.7 %
Joint 6	-	−6.7 %	−6.0 %

References

[1] Pan, Z., Polden, J., Larkin, N., Duin, S.V., Norrish, J.. Recent progress on programming methods for industrial robots. Robotics and Computer-Integrated Manufacturing 2012;28(2):87 – 94.
[2] Bukchin, J., Tzur, M.. Design of flexible assembly line to minimize equipment cost. IIE Transactions 2000;32(7):585–598.
[3] LaValle, S.M., Kuffner Jr, J.J.. Randomized kinodynamic planning. In: Robotics and Automation, 1999 IEEE International Conference on. 1999,.
[4] Bohlin, R., Kavraki, L.E.. Path planning using lazy PRM. In: IEEE International Conference on Robotics and Automation. 2000, p. 521–528.
[5] LaValle, S., Hutchinson, S.. Optimal motion planning for multiple robots having independent goals. Robotics and Automation, IEEE Transactions on 1998;14(6):912–925.
[6] Spensieri, D., Bohlin, R., Carlson, J.S.. Coordination of robot paths for cycle time minimization. Automation Science and Engineering (CASE), 2013 IEEE International Conference on 2013;:522–527.
[7] Segeborn, J., Segerdahl, D., Carlson, J.S., Ekstedt, F., Carlsson, A., Söderberg, R.. A generalized method for weld load balancing in multi station sheet metal assembly lines. 2011,Proceedings of the ASME 2011 International Mechanical Engineering Congress & Exposition, Denver, Colorado, USA, November 11-17.
[8] Karaman, S., Frazzoli, E.. Sampling-based algorithms for optimal motion planning. The International Journal of Robotics Research 2011;30(7):846–894.
[9] Gleeson, D., Björkenstam, S., Bohlin, R., Carlson, J.S., Lennartson, B.. Energy efficient and collision free motion of industrial robots using optimal control. Automation Science and Engineering (CASE), 2015 IEEE International Conference on 2015;:495–500.
[10] Björkenstam, S., Gleeson, D., Bohlin, R., Carlson, J.S., Lennartson, B.. Energy efficient and collision free motion of industrial robots using optimal control. Automation Science and Engineering (CASE), 2013 IEEE International Conference on 2013;:510–515.
[11] Wächter, A., Biegler, L.T.. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. Mathematical Programming 2006;106(1):25–57.
[12] Ustyan, T., Jönsson, V.. Implementation of a generic virtual robot controller. Master's thesis; Chalmers University of Technology; 2011.
[13] Forsman, D.. Bangenerering för industrirobot med 6 frihetsgrader. Master's thesis; Tekniska Högskolan i Linköping; 2004.
[14] Featherstone, R.. Rigid body dynamics algorithms; vol. 49. Springer Berlin:; 2008.
[15] Riazi, S., Bengtsson, K., Wigstrom, O., Vidarsson, E., Lennartson, B.. Energy optimization of multi-robot systems. In: Automation Science and Engineering (CASE), 2015 IEEE International Conference on. 2015, p. 1345–1350.
[16] Paes, K., Dewulf, W., Elst, K.V., Kellens, K., Slaets, P.. Energy efficient trajectories for an industrial ABB robot. Procedia CIRP 2014;15:105 – 110. 21st CIRP Conference on Life Cycle Engineering.
[17] Gerdts, M., Henrion, R., Hömberg, D., Landry, C.. Path planning and collision avoidance for robots. Numerical Algebra, Control and Optimization 2012;2(3):437–463.